```
1
2  #! /usr/bin/env python3
3
4
5  ### Muli-Step Tactile Sensor Reading Test Script ###
6
7  ###
8  # KINOVA (R) KORTEX (TM)
9  #
10 # Copyright (c) 2018 Kinova inc. All rights reserved.
11 #
12 # This software may be modified and distributed
13 # under the terms of the BSD 3-Clause license.
14 #
15 # Refer to the LICENSE file for details.
16 #
17 ###
18
19 from sunau import AUDIO_UNKNOWN_SIZE
20 import sys
21 import os
22 import time
23 import threading
24 import numbers
25
26 from kortex_api.autogen.client_stubs.BaseClientRpc import BaseClient
27 from kortex_api.autogen.client_stubs.BaseCyclicClientRpc import BaseCyclicClient
28
29 from kortex_api.autogen.messages import Base_pb2, BaseCyclic_pb2, Common_pb2
30
31 # DEFINITIONS:
32
33 # Index Variables
34 # Initialize the varibales for the x,y and z slots of the soft and hard
35 # list.
36 x = 0;
37 y = 1;
38 z = 2;
39
40 # Sensor Dimensions - Unit: cm
41 height = 11.5;
42 diameter = 6.5;
43 radius = diameter/2;
44
45
46 # Soft Inclusion Phantom Center Coordinates - Unit: cm
47 soft = [45, 5, -1.5];
48
49 # Subtracting the radius of the sample tube so the center of the
50 # sample tube sits above the center of the phantom.
51 softx = soft[x] - radius
52 softy = soft[y]
53
54 # Subtracting the height of the sample tube so the edge of the
55 # sample tube hovers above the phantom.
56 softz = soft[z] + height
57 print(softx, softy, softz)
58
59 # Hard Inclusion Phantom Center Coordinates - Unit: cm
```

```
hard =  [45, -5, -0.5];
hardx = hard[x] - radius
hardy = hard[y]
hardz = hard[z] + height
print(hardx, hardy, hardz)

# End Effector Angle - Unit: degrees
# These angles are choosen to fix the end-effector of the Kinova arm at
# 90-degree angle.
adx = 90;
ady = 0;
adz = 90;

# Pyrex Coordinates - Unit: cm
P1 = [47, 10, -3.5]; #Top Left
P2 = [47, 10, -3.5]; #Top Right
P3 = [30, -14, -3.5]; #Bottom Right
P4 = [30, 10, -3.5]; #Bottom Left

# Maximum allowed waiting time during actions (in seconds)
TIMEOUT_DURATION = 1000

# Create closure to set an event after an END or an ABORT
# Takes an input of "e" to see if the progam has finished or aborted.
# Returns the notification of the event that occured.

def check_for_end_or_abort(e):
    """Return a closure checking for END or ABORT notifications
    Arguments:
    e -- event to signal when the action is completed
        (will be set when an END or ABORT occurs)
    """
    def check(notification, e = e):
        print("EVENT : " + \
              Base_pb2.ActionEvent.Name(notification.action_event))
        if notification.action_event == Base_pb2.ACTION_END \
        or notification.action_event == Base_pb2.ACTION_ABORT:
            e.set()
    return check

def example_test_movement(base, base_cyclic):

    print("Starting Cartesian action movement ...")
    action = Base_pb2.Action()
    action.name = "Example Cartesian action movement"
    action.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action.reach_pose.target_pose
    cartesian_pose.x = feedback.base.tool_pose_x          # (meters)
    cartesian_pose.y = feedback.base.tool_pose_y     # (meters)
    cartesian_pose.z = feedback.base.tool_pose_z     # (meters)
    cartesian_pose.theta_x = feedback.base.tool_pose_theta_x # (degrees)
    cartesian_pose.theta_y = feedback.base.tool_pose_theta_y # (degrees)
    cartesian_pose.theta_z = feedback.base.tool_pose_theta_z # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
```

```
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)


    # Soft Movement 1
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = softx/100        # (meters)
    cartesian_pose.y = softy/100       # (meters)
    cartesian_pose.z = softz/100 + 15/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

    # Soft Movement 2
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = softx/100        # (meters)
    cartesian_pose.y = softy/100       # (meters)
    cartesian_pose.z = softz/100         # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
```

```
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Soft Movement 3
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = softx/100       # (meters)
    cartesian_pose.y = softy/100             # (meters)
    cartesian_pose.z = softz/100 - 0.2/100       # (meters)
    cartesian_pose.theta_x = adx     # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Soft Movement 4
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = softx/100        # (meters)
```

```
    cartesian_pose.y = softy/100       # (meters)
    cartesian_pose.z = softz/100 - 0.4/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Soft Movement 5
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = softx/100        # (meters)
    cartesian_pose.y = softy/100       # (meters)
    cartesian_pose.z = softz/100 - 0.6/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Soft Movement 6
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
```

```
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = softx/100        # (meters)
    cartesian_pose.y = softy/100       # (meters)
    cartesian_pose.z = softz/100 - 0.8/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Soft Movement 7
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = softx/100        # (meters)
    cartesian_pose.y = softy/100       # (meters)
    cartesian_pose.z = softz/100 - 2/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Retracting Arm
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""
```

```
353
354     feedback = base_cyclic.RefreshFeedback()
355
356     cartesian_pose = action1.reach_pose.target_pose
357     # These are loaded directly from Kinova Web app (the first three are divided by
    100 since it gives them in mm)
358
359     cartesian_pose.x = softx/100        # (meters)
360     cartesian_pose.y = softy/100       # (meters)
361     cartesian_pose.z = softz/100 + 15/100       # (meters)
362     cartesian_pose.theta_x = adx   # (degrees)
363     cartesian_pose.theta_y = ady     # (degrees)
364     cartesian_pose.theta_z = adz     # (degrees)
365
366     e = threading.Event()
367     notification_handle = base.OnNotificationActionTopic(
368         check_for_end_or_abort(e),
369         Base_pb2.NotificationOptions()
370     )
371
372     print("Executing action")
373     base.ExecuteAction(action1)
374
375     print("Waiting for movement to finish ...")
376     finished = e.wait(TIMEOUT_DURATION)
377     base.Unsubscribe(notification_handle)
378
379 # Transition to Hard Inclusion
380     print("Starting Cartesian action movement ...")
381     action1 = Base_pb2.Action()
382     action1.name = "Movement 1- Top Left"
383     action1.application_data = ""
384
385     feedback = base_cyclic.RefreshFeedback()
386
387     cartesian_pose = action1.reach_pose.target_pose
388     # These are loaded directly from Kinova Web app (the first three are divided by
    100 since it gives them in mm)
389
390     cartesian_pose.x = hardx/100        # (meters)
391     cartesian_pose.y = hardy/100       # (meters)
392     cartesian_pose.z = hardz/100 + 15/100       # (meters)
393     cartesian_pose.theta_x = adx   # (degrees)
394     cartesian_pose.theta_y = ady     # (degrees)
395     cartesian_pose.theta_z = adz     # (degrees)
396
397     e = threading.Event()
398     notification_handle = base.OnNotificationActionTopic(
399         check_for_end_or_abort(e),
400         Base_pb2.NotificationOptions()
401     )
402
403     print("Executing action")
404     base.ExecuteAction(action1)
405
406     print("Waiting for movement to finish ...")
407     finished = e.wait(TIMEOUT_DURATION)
408     base.Unsubscribe(notification_handle)
409
410 # Hard Movement 1
```

```
411     print("Starting Cartesian action movement ...")
412     action1 = Base_pb2.Action()
413     action1.name = "Movement 1- Top Left"
414     action1.application_data = ""
415 
416     feedback = base_cyclic.RefreshFeedback()
417 
418     cartesian_pose = action1.reach_pose.target_pose
419     # These are loaded directly from Kinova Web app (the first three are divided by
    100 since it gives them in mm)
420 
421     cartesian_pose.x = hardx/100        # (meters)
422     cartesian_pose.y = hardy/100       # (meters)
423     cartesian_pose.z = hardz/100         # (meters)
424     cartesian_pose.theta_x = adx   # (degrees)
425     cartesian_pose.theta_y = ady     # (degrees)
426     cartesian_pose.theta_z = adz     # (degrees)
427 
428     e = threading.Event()
429     notification_handle = base.OnNotificationActionTopic(
430         check_for_end_or_abort(e),
431         Base_pb2.NotificationOptions()
432     )
433 
434     print("Executing action")
435     base.ExecuteAction(action1)
436 
437     print("Waiting for movement to finish ...")
438     finished = e.wait(TIMEOUT_DURATION)
439     base.Unsubscribe(notification_handle)
440 
441 # Hard Movement 2
442     print("Starting Cartesian action movement ...")
443     action1 = Base_pb2.Action()
444     action1.name = "Movement 1- Top Left"
445     action1.application_data = ""
446 
447     feedback = base_cyclic.RefreshFeedback()
448 
449     cartesian_pose = action1.reach_pose.target_pose
450     # These are loaded directly from Kinova Web app (the first three are divided by
    100 since it gives them in mm)
451 
452     cartesian_pose.x = hardx/100       # (meters)
453     cartesian_pose.y = hardy/100             # (meters)
454     cartesian_pose.z = hardz/100 - 0.2/100       # (meters)
455     cartesian_pose.theta_x = adx     # (degrees)
456     cartesian_pose.theta_y = ady     # (degrees)
457     cartesian_pose.theta_z = adz     # (degrees)
458 
459     e = threading.Event()
460     notification_handle = base.OnNotificationActionTopic(
461         check_for_end_or_abort(e),
462         Base_pb2.NotificationOptions()
463     )
464 
465     print("Executing action")
466     base.ExecuteAction(action1)
467 
468     print("Waiting for movement to finish ...")
```

```
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Soft Movement 3
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = hardx/100        # (meters)
    cartesian_pose.y = hardy/100       # (meters)
    cartesian_pose.z = hardz/100 - 0.4/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Hard Movement 4
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = hardx/100        # (meters)
    cartesian_pose.y = hardy/100       # (meters)
    cartesian_pose.z = hardz/100 - 0.6/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

```

```
    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Hard Movement 5
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = hardx/100        # (meters)
    cartesian_pose.y = hardy/100       # (meters)
    cartesian_pose.z = hardz/100 - 0.8/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# hard Movement 6
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = hardx/100        # (meters)
    cartesian_pose.y = hardy/100       # (meters)
    cartesian_pose.z = hardz/100 -2/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
```

```
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

# Retracting Arm
    print("Starting Cartesian action movement ...")
    action1 = Base_pb2.Action()
    action1.name = "Movement 1- Top Left"
    action1.application_data = ""

    feedback = base_cyclic.RefreshFeedback()

    cartesian_pose = action1.reach_pose.target_pose
    # These are loaded directly from Kinova Web app (the first three are divided by 100 since it gives them in mm)

    cartesian_pose.x = hardx/100        # (meters)
    cartesian_pose.y = hardy/100       # (meters)
    cartesian_pose.z = hardz/100 + 15/100       # (meters)
    cartesian_pose.theta_x = adx   # (degrees)
    cartesian_pose.theta_y = ady     # (degrees)
    cartesian_pose.theta_z = adz     # (degrees)

    e = threading.Event()
    notification_handle = base.OnNotificationActionTopic(
        check_for_end_or_abort(e),
        Base_pb2.NotificationOptions()
    )

    print("Executing action")
    base.ExecuteAction(action1)

    print("Waiting for movement to finish ...")
    finished = e.wait(TIMEOUT_DURATION)
    base.Unsubscribe(notification_handle)

    if finished:
        print("Cartesian movement completed")
    else:
        print("Timeout on action notification wait")
    return finished

def main():

    # Import the utilities helper module
    sys.path.insert(0, os.path.join(os.path.dirname(__file__), ".."))
    import utilities

    # Parse arguments
    args = utilities.parseConnectionArguments()

    # Create connection to the device and get the router
    with utilities.DeviceConnection.createTcpConnection(args) as router:
```

```
644
645         # Create required services
646         base = BaseClient(router)
647         base_cyclic = BaseCyclicClient(router)
648
649         # Example core
650         success = True
651         success &= example_test_movement(base, base_cyclic)
652         #success &= example_test_movement(base, base_cyclic)
653
654         return 0 if success else 1
655
656 if __name__ == "__main__":
657     exit(main())
658
```